
Software
Engineering 1
Software
Engineering 2
Software
Engineering 3

WELCOME

Domain Science & Engineering
A Precursor for Requirements Engineering

FM 2012 Tutorial, CNAM, 28 August 2012

Dines Bjørner

DTU Informatics

July 31, 2012: 09:02

Begin of Lecture 1: First Session — Introduction

Domains, TripTych, Issues

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012

Tutorial Schedule

- **Lectures 1–2** 9:00–9:40 + 9:50–10:30

√ 1 **Introduction** **Slides 1–35**

2 **Endurant Entities: Parts** Slides 36–110

- **Lectures 3–5** 11:00–11:15 + 11:20–11:45 + 11:50–12:30

3 **Endurant Entities: Materials, States** Slides 111–142

4 **Perdurant Entities: Actions and Events** Slides 143–174

5 **Perdurant Entities: Behaviours** Slides 175–285

Lunch **12:30–14:00**

- **Lectures 6–7** 14:00–14:40 + 14:50–15:30

6 **A Calculus: Analysers, Parts and Materials** Slides 286–339

7 **A Calculus: Function Signatures and Laws** Slides 340–377

- **Lecturs 8–9** 16:00–16:40 + 16:50–17:30

8 **Domain and Interface Requirements** Slides 378–424

9 **Conclusion: Comparison to Other Work** Slides 428–460

Conclusion: What Have We Achieved Slides 425–427 + 461–472

Summary

- This tutorial covers
  - a **new science & engineering of domains** as well as
  - a **new foundation for software development**

  We treat the latter first.
- Instead of commencing with requirements engineering,
  - whose pursuit may involve repeated,
  - but unstructured forms of domain analysis,
  - we propose a predecessor phase of domain engineering.
- That is, we single out domain analysis as an activity to be pursued prior to requirements engineering.

- In emphasising **domain engineering** as a predecessor phase
  - we, at the same time, introduce a number of facets
  - that are **not present,** we think,
  - in current software engineering studies and practices.
- **One facet** is **the construction of separate domain descriptions.**
  - Domain descriptions are void of any reference to requirements
  - and encompass the **modelling** of **domain phenomena**
  - without regard to their being computable.

- **Another facet** is **the pursuit of domain descriptions as a free-standing activity.**
  - In this tutorial we emphasize domain description development need not lead to [further] software development.
  - This gives a new meaning to business process engineering, and should lead to
    - a deeper understanding of a domain
    - and to possible non-IT related business process re-engineering of areas of that domain.
- In this tutorial we shall investigate
  - a method for analysing domains,
  - for constructing domain descriptions
  - and some emerging scientific bases.

- **Our contribution** to **domain analysis** is that we view domains as having the following **ontology**.
  - There are the **entities** that we can describe and then there is "the rest" which we leave un-described.
  - We **analyse** entities into
    - **endurant entities** (Slides 51–134) and
    - **perdurant entities** (Slides 144–280).

    that is
    - **parts** and **materials** as **endurant entities** (Slides 51–134) and
    - **discrete actions**, **discrete events** and **behaviour s** as **perdurant entities** (Slides 146–280), respectively.
- Another way of looking at **entities** is as
  - **discrete entities** (Slides 51–174 and 176–245),
  - **continuous entities** (Slides 112–134 and Slides 246–280).

- We also contribute to the **analysis** of **discrete endurants** in terms of the following notions:
  - **part types** and **material types** (Slides 51–69 and Slides 112–134),
  - **part unique identifiers** (Slides 75–77),
  - **part mereology** (Slides 78–93) and
  - **part attributes** and **material attributes** (Slides 94–105, Slides 121–125) and
  - **material laws** (Slides 126–131).
- Of the above we point to the introduction, into **computing science** and **software engineering** of the notions of
  - **materials** (Slides 112–134) and
  - **continuous behaviours** (Slides 246–280)

  as novel.

1. Introduction

- I remind You of the **abstract**,
  - Slide 7,
  - as for the **contributions** of this tutorial.
- This is primarily a **methodology** paper.
- By a **method** we shall understand
  - a set of **principles**
  - for **selecting** and **applying**
  - a number of **techniques** and **tools**
  - in order to **analyse** a **problem**
  - and **construct** an **artefact**.
- By **methodology** we shall understand
  - the study and knowledge about methods.

- This tutorial contributes to
  - the study and knowledge
  - of software engineering development methods.
- Its contributions lie in by suggesting and exploring
  - **domain engineering** and
  - domain engineering as a basis for **requirements engineering**.
- We are not saying
  - *"thou must develop software this way"*,
- but we do suggest
  - that since it is possible
  - and makes sense to do so
  - it may also be wise to do so.

1.1. Domains: Some Definitions

- By a **domain** we shall here understand
  - an area of human activity
  - characterised by observable phenomena:
    - **entities**
      - whether **endurants** (manifest **parts** and **materials**)
      - or **perdurants** (**actions**, **events** or **behaviours**),
    - whether
      - **discrete**
      - **continuous**;
    - and of their **properties**;

Example: 1 Some Domains. Some examples are:

air traffic,
airport,
banking,
consumer market,
container lines,
fish industry,
health care,
logistics,
manufacturing,
pipelines,
securities trading,
transportation
etcetera. ■

1.1.1. Domain Analysis

- By **domain analysis** we shall understand
  - an inquiry into the domain,
  - its **entities**
    - (manifest **parts** and **materials**;
    - **actions**, **events** and **behaviours**)
  - and their **properties**

Example: 2 A Container Line Analysis.

We omit enumerating entity properties.

- *parts:*
  - container,
  - vessel,
  - terminal port, etc.;
- *actions:*
  - container loading,
  - container unloading,
  - vessel arrival in port, etc.;
- *events:*
  - container falling overboard;
  - container afire;
  - etc.;
- *behaviour:*
  - vessel voyage,
  - across the seas,
  - visiting ports, etc.

Length of a container is a container *property*.
Name of a vessel is a vessel *property*.
Location of a container terminal port is a port *property*.

1.1.2. Domain Descriptions

- By a **domain description** we shall understand
  - a **narrative description**
  - tightly coupled (say line-number-by-line-number)
  - to a **formal description**.
- To develop a **domain description**
  requires a thorough amount of **domain analysis**.

Example: 3 A Transport Domain Description.

- *Narrative:*
  - a transport net, n:N,
    consists of an aggregation of hubs, hs:HS,
    which we "concretise" as a set of hubs, H-**set**, and
    a set of links, ls:LS, respectively L-**set**,
- *Formalisation:*

type N, HS, LS, Hs = H-**set**, Ls = L-**set**, H, L
value
  obs_HS: N→HS,
  obs_LS: N→LS.
  obs_Hs: N→H-**set**,
  obs_Ls: N→L-**set**.

1.1.3. Domain Engineering

- By **domain engineering** we shall understand
  - the **engineering** of a domain description,
  - that is,
    - the rigorous construction of domain descriptions, and
    - the further analysis of these, creating **theories of domains**[1], etc.

[1] Examples of such theories, albeit in rather rough forms, are given in Appendices B–C.

- The size[2], structure[3] and complexity[4] of interesting domain descriptions is usually such as to put a special emphasis on engineering:
  - the management and organisation of several, typically 5–6 collaborating domain describers,
  - the ongoing check of description quality, completeness and consistency, etcetera.

[2] usually, say a hundred pages
[3] usually a finely sectioned document of may subsub···subsections
[4] having many cross-references between subsub···subsections

1.1.4. Domain Science

- By **domain science** we shall understand
  - two things:
    - the general study and knowledge of
      - how to create and handle domain descriptions
      - (a general theory of domain descriptions)

      and
    - the specific study and knowledge of a particular domain.
  - The two studies intertwine.

1.2. The Triptych of Software Development

- We suggest a "dogma":
  - before **software** can be **designed** one must understand[5] the **requirements**; and
  - before **requirements** can be expressed one must understand[6] the **domain**.
- We can therefore view software development as ideally proceeding in three (i.e., **TripTych**) phases:
  - an initial phase of **domain engineering**, followed by
  - a phase of **requirements engineering**, ended by
  - a phase of **software design**.

[5] Or maybe just: have a reasonably firm grasp of
[6] See previous footnote!

- In the **domain engineering phase** ($\mathcal{D}$)
  - a domain is analysed, described and "theorised",
  - that is, the beginnings of a **specific domain theory** is established.
- In the **requirements engineering phase** ($\mathcal{R}$)
  - a **requirements prescription** is constructed —
  - significant fragments of which are "derived",
  - systematically, from the **domain description**.
- In the **software design phase** ($\mathcal{S}$)
  - a **software design**
  - is derived, systematically, rigorously or formally,
  - from the **requirements prescription**.
- Finally the $\mathcal{S}$oftware is proven correct with respect to the $\mathcal{R}$equirements under assumption of the $\mathcal{D}$omain: $\mathcal{D}, \mathcal{S} \models \mathcal{R}$.

- By a **machine** we shall understand the **hardware** and **software** of a target, i.e., a required **IT system**.
- In [dines:ugo65:2008,psi2009,Kiev:2010ptI] we indicate how one can "derive" significant parts of requirements from a suitably comprehensive domain description – basically as follows.
  - **Domain projection**: from a domain description one **project**s those areas that are to be somehow manifested in the software.
  - **Domain initialisation**: for that resulting projected requirements prescription one **initialise**s a number of part types as well as action and behaviour definitions, from less **abstract** to more **concrete**, specific types, respectively definitions.

- ◈ **Domain determination**: hand-in-hand with domain initialisation a[n interleaved] stage of making values of types less **non-deterministic**, i.e., more **deterministic**, can take place.
- ◈ **Domain extension**: Requirements often arise in the context of new business processes or technologies either placing old or replacing human processes in the domain. Domain extension is now the 'enrichment' of the domain requirements, so far developed, with the description of these new business processes or technologies.
- ◈ Etcetera.

• The result of this part of "**requirements derivation**" is the **domain requirements**.

- A set of domain-to-requirements operators similarly exists for constructing **interface requirements**
  - from the domain description and,
  - independently, also from knowledge of the **machine**
  - for which the required IT system is to be developed.
- We illustrate the **techniques** of **domain requirements** and **interface requirements** in Sect. 4.
- Finally **machine requirements** are "derived"
  - from just the knowledge of the **machine**,
  - that is,
    - the target hardware and
    - the software system tools for that hardware.

- When you review this section ('A Triptych of Software Development')
  - then you will observe how 'the domain'
  - predicates both the requirements
  - and the software design.
- For a specific domain one may develop
  - many (thus related) requirements
  - and from each such (set of) requirements
  - one may develop many software designs.
- We may characterise this multitude of domain-predicated requirements and designs as a **product line** [dines-maurer].
- You may also characterise domain-specific developments as representing another 'definition' of **domain engineering**.

1.3. Issues of Domain Science & Engineering

- We specifically focus on the following issues of domain science &[7] engineering:
  - (i) which are the "things" to be described[8],
  - (ii) how to analyse these "things" into description structures[9],
  - (iii) how to describe these "things" informally and formally,
  - (iv) how to further structure descriptions[10], and a further study of
  - (v) **mereology**[11].

[7] When we put '&' between two terms that the compound term forms a whole concept.

[8] **endurants** [manifest entities henceforth called **parts** and **materials**] and **perdurants** [**actions**, **events**, **behaviours**]

[9] **atomic** and **composite**, **unique identifiers**, **mereology**, **attributes**

[10] *intrinsics*, *support technology*, *rules & regulations*, *organisation & management*, *human behaviour* etc.

[11] the study and knowledge of parts and relations of parts to other parts and a "whole".

1.4. Structure of Paper

- It is always a good idea to consult and study the *table of contents* listing of the colloquium one is listening to. Therefore one is brought here:

- First (Sect. 1) we introduce the problem. And that was done above.

- Then, in (Sects. 2–10)
  - we bring a rather careful analysis of
  - the concept of the observable, manifest phenomena
  - that we shall refer to as entities
- We strongly think that these sections of this tutorial
  - brings a simple, to our taste, simple and elegant
  - reformulation of what is usually called *"data modelling"*,
  - in this case for domains —
  - but with major aspects applicable as well to
  - requirements development and software design.

- That analysis focuses on
  - **endurant entities**, also called **parts** and **materials**,
    - those that can be observed at no matter what time,
    - i.e., entities of substance or continuant, and
  - **perdurant entities**: **action**, **event** and **behaviour** entities, those
    - that occur,
    - that happen,
    - that, in a sense, are accidents.

- **We think** that "decomposition" of the "data analysis" problem into
  - discrete parts and continuous materials,
  - atomic and composite parts,
  - their unique identifiers and mereology, and
  - their attributes
  - **is novel**,
  - and differs from past practices in domain analysis.

- In Sect. 11 we suggest
  - for each of the entity categories
    - parts,
    - materials,
    - actions,
    - events and
    - behaviours,
  - a calculus of meta-functions:
    - analytic functions,
      - that guide the domain description developer
      - in the process of selection,

      and
    - so-called discovery functions,
      - that guide that person
      - in "generating" appropriate domain description texts, informal and formal.

- The domain description calculus is to be thought of
  - as directives to the domain engineer,
  - mental aids that help a team of domain engineers
  - to steer it simply through the otherwise daunting task
  - of constructing a usually large domain description.
- Think of the calculus
  - as directing
  - a human calculation
  - of domain descriptions.
- Finally the domain description calculus section
  - suggests a number of laws that the
  - domain description process ought satisfy.

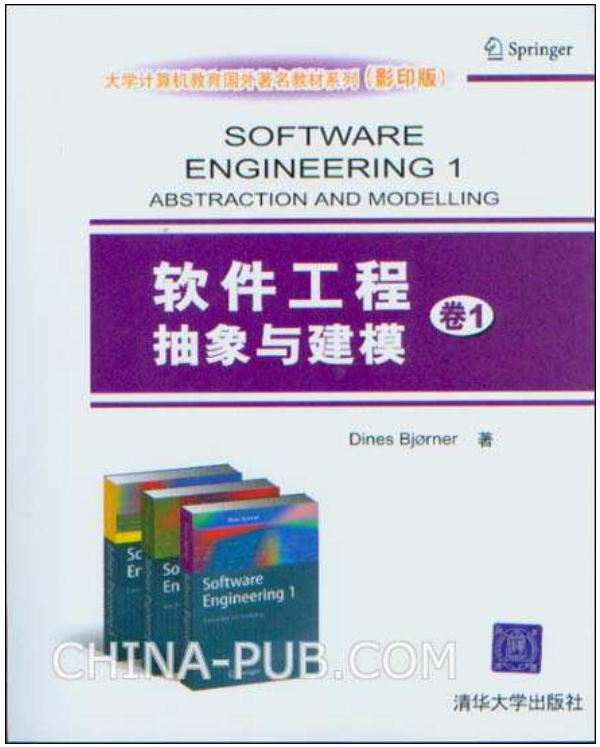
- Finally (Sect. 12) we bring a brief survey of the kind of requirements engineering
  - that one can now pursue based on a reasonably comprehensive domain description.
  - We show how one can systematically, but not automatically
  - "derive" significant fragments
    - of requirements prescriptions
    - from domain descriptions.

- The formal descriptions will here be expressed in the RAISE [RaiseMethod] Specification Language, RSL.
- We otherwise refer to [TheSEBook1wo].
- Appendix brings a short primer,
  mostly on the syntactic aspects of RSL.
- But other model-oriented formal specification languages can be used with equal success; for example:
  - Alloy [alloy],
  - Event B [JRAbrial:TheBBooks],
  - VDM [e:db:Bj78bwo,e:db:Bj82b,jf-pgl-97] and
  - Z [m:z:jd+jcppw96].

End of Lecture 1: First Session — Introduction

Domains, TripTych, Issues

FM 2012 Tutorial, Dines Bjørner, Paris, 28 August 2012



SHORT BREAK